CS-336 Project 2015: Hardware Parts

Matthew Eder
Rutgers University
Piscataway, NJ, USA
Email: mre56@scarletmail.rutgers.edu

Michael Azzimaro
Rutgers University
Piscataway, NJ, USA
Email: michazzi@scarletmail.rutgers.edu

Michael Calabrese
Rutgers University
Piscataway, NJ, USA
Email: mrcalabrese2@gmail.com

***Abstract*— In this project, we intend to build a computer hardware Data Base Management System from the ground up that composes of multiple stages. The database system will be used for the sole purpose of immersing its users with computer hardware data. The overall project begins with stage one, which is intended for analyzing how specific users of the computer hardware database interact with the database environment. Stage two will bring stage one to life through a use of a pictorial design of the computer hardware database system. Here, an ER model will be presented as a means of describing relations between entities of the computer hardware database system. Also stated in this report is stage three, describing how we intend to lay out our database design through the use of SQL tables. Finally, stage four will describe the User Interface that will be based off of the computer hardware database system design. Overall, this project seeks to build a computer hardware database system for multiple user types. We intend to design and implement a computer hardware database system that encourages easy accessibility and maneuverability for its users.**

A. Stage1 - The Requirement Gathering Stage.

- The general system description: This is a database of computer hardware parts and detailed information about those parts. The goal of this is to simplify part comparisons. Parts have multiple filterable categories to allow easy and quick access when comparing a large number of parts. By acting as a selection guide this database could be used to assist people to build there own computers. There are four different types of users in this database manufactures, sellers, account users and guest users.
- The 4 types of users (grouped by their data access/update rights):
  - The user's access rights: The four types of users for our database include the manufacturer, the seller, account customers and guest users. The manufacturers main purpose is to update the database, detailing new product information as well as release dates on their respective computer hardware. Out of all users of the database, manufacturers are allowed the most data access and update rights. Specifically, manufactures are permitted to add new data to the database as well as remove previously added information respectively. Different manufactures cannot alter computer hardware information that is not their own. Additionally, manufacturers are allowed read access to account-customer reviews.

    The sellers will compose of a variety of different online retailers for computer hardware parts. The seller's main contribution to the database is to list product prices as well as provide computer hardware specifications that are provided by the manufacturer. The seller will be allowed data access to all manufacture products. Sellers will also be allowed to group hardware parts into categories based on its purpose and functionality. As a way of receiving feedback from customers, sellers will also be granted the ability to create a customer reviews section on the respective product page. As compared to manufactures, sellers are limited when it comes to update rights. No seller is permitted to alter computer hardware product information unless authorized by the manufacturer themselves. Additionally, sellers are not allowed to modify and/or delete account-customer reviews unless the content of the review pertains no correspondence to the product whatsoever.

    Account users can access the prices of a specific grouping of hardware pieces, they can also get seller information based on location and price. Account users also gain read access to the manufacturer information and the specifications of the products. They also gain read and write access to reviews of those products with the ability to rate and write reviews.

    Guest users on the other hand maintain most of the same access rights as account users, however they cannot write reviews on hardware products. Guest users also lack the ability to choose which seller they want to purchase from, while account users have this ability. Both types of users lack the ability to add and remove products from the database, as well as manipulate any information on the products other than reviews.
  - The real world scenarios:

    * Scenario1 description: A manufacture has come out with a new line of solid-state drives and wants to add one to the database, along with information about the part.
    * System Data Input for Scenario1:
      Add a new SDD along with product information

INSERT INTO HardwareDB(PartName, type, Capacity, Interface, Form Factor, SeqReadm SeqWrite, RandRead, RandWirite, weight)
VALUES('Crucial MX100 CT256MX100SSD', 'SSD', 256, 'SATA III', 2.5, 550, 330, 85000, 70000, 0.25)

* Input Data Types for Scenario1:
Char[], char[], int, char[], float, int, int, int, int, float
* System Data Output for Scenario1:
A message that shows if the insertion was successful or not
* Output Data Types for Scenario1:
char[]

* Scenario2 description: A manufacture has realized that some information about their product is incorrect and needs to correct to it. A monitor QX2711 has a DVI-I connection instead of DVI-D connection.
* System Data Input for Scenario2:
Change the interface attribute of the monitor QX2711 to show there is one DVI-I connection
UPDATE HardwareDB
SET Interface=1xDVI-I
WHERE name='QX2711';
* Input Data Types for Scenario2:
char[], char[]
* System Data Output for Scenario2:
A message that shows if the update successful or not
* Output Data Types for Scenario2:
char[]

* Scenario3 description: There are multiple real world scenarios a seller user type can act out within the database environment. For example: A seller user type can submit a request to update price information on a computer hardware product in their inventory. The seller will then receive approval or disapproval from the manufacturer, deciding whether or not the requested price alteration can be accepted. Product standards are constantly being updated and improved as new merchandise is released each year. With such adjustments, constant alteration in price is required to keep customer satisfaction at a high rate. Sellers cannot afford the risk of keeping product prices static. A lack of constant updating of price information can lead retailers loosing sales and future customers. Additionally, negative customer reviews can erupt from such an occurrence that can most definitely impact both the seller and manufacturer for the worst. Such a scenario is very typical amongst these two user types and should be accommodated for in the computer hardware database. Overall, the interaction between seller and manufacture is necessary when regarding price modifications that ultimately can lead to a beneficial relationship between the two when sharing a database system.
* System Data Input for Scenario3: Sellers will input the updated price of the hardware product into the database. Seller users must interact with manufacturers and the database system constantly with price alterations in mind in order to keep up with the hardware standards of the time.
* Input Data Types for Scenario3: The input data type for this scenario is a float data type since sellers are updating money values in the database system.
* System Data Output for Scenario3: After the seller has inputted the requested price alteration, the manufacturer will then reply to that request as output to the seller. This consists of either an approval or disapproval answer by the manufacturer.
* Output Data Types for Scenario3: The output data type for this scenario is a char data type considering the fact that the seller will receive output as either an approval or disapproval.

* Scenario4 description: Another real world scenario a seller user type can act out within the database environment involves the interaction between seller and account-only customers. As compared to guest users, account customers are allowed increased access to seller and product information. For example: A seller user type can add new products to their inventory. More inventory has a probable chance to lead to more potential customers. If an account customer decides to buy a product a seller is retailing, the seller will receive the amounted price on the specific item from the customer. This retailer-customer relationship and interaction is necessary in order to keep business operations flowing at a normal rate. If the connection between seller and account-customer is not made through the description of well defined products, the seller user type will find it very difficult to keep its business in check and ultimately loose ties with customers and manufactures as a whole. By constantly adding and updating information by interacting with the computer hardware database, sellers can easily access their inventory while account-customers can simply read, redeem and rate said inventory, leading to future healthy connections between seller and account-customers.
* System Data Input for Scenario4: Sellers input product descriptions and details as well as shipping costs into the database system under an item description attribute. Overall, each seller will provide product descriptions, prices and shipping costs as a way to gain potential revenue from

customers and enhance sales.

* Input Data Types for Scenario4: The input data type for this scenario is a char data type. This is due to the fact that the seller will input product descriptions for each item in the computer hardware database.
* System Data Output for Scenario4: After item descriptions and details have been inputted into the database, sellers will hope receive output in the form of sales from customers.
* Output Data Types for Scenario4: The output data type for this scenario is a float data type. Retailers listed in the database system will be sent output in the form of currency, making the float data type the definitive output type.

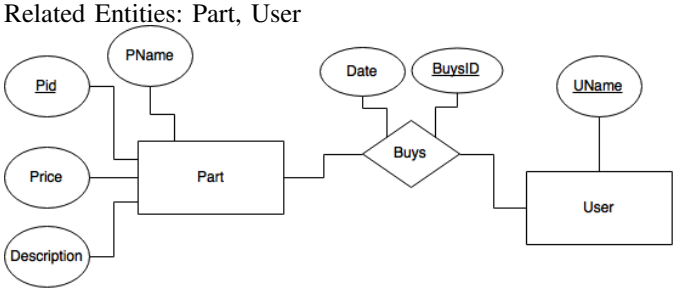
* Scenario5 description: An account user purchased an item from the database and now wishes to write a review of the product in the database.
* System Data Input for Scenario5: number of stars, short reason
* Input Data Types for Scenario5: int, char[]
* System Data Output for Scenario5: confirmation message
* Output Data Types for Scenario5: char[]

* Scenario6 description: An account user queries the database for a GPU, and receives a list of parts. From those parts he can query based on price, seller, manufacturer, ratings or name. after selecting the product, if there is an option for multiple sellers the user gains access to select particular sellers for the product.
* System Data Input for Scenario6: video card, price <300 dollars
* Input Data Types for Scenario6: char[], int
* System Data Output for Scenario6: direct link to product on web
* Output Data Types for Scenario6: char[]

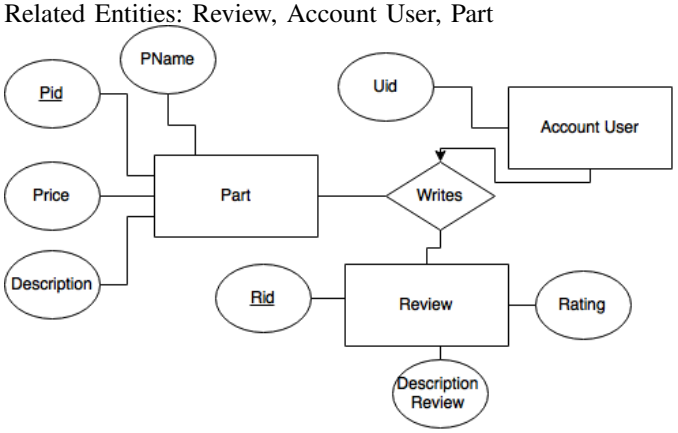
* Scenario7 description: A guest user wants to purchase a product, so he uses the database to find the best fit for his restrictions. Guest users, however, can not choose which seller they want if their are multiple options. They also have less options in terms of filtering results, only by price, review rating and name.
* System Data Input for Scenario7: name of product, price, rating
* Input Data Types for Scenario7: char[], float, int
* System Data Output for Scenario7: Direct link to product on the web
* Output Data Types for Scenario7: char[]
* Scenario8 description: A guest user wants to read a review about a specific part. They will query the database with the part name and receive a table of reviews.
* System Data Input for Scenario8: name of product
* Input Data Types for Scenario8: char[]
* System Data Output for Scenario8: list of reviews
* Output Data Types for Scenario8: char[]

B. Stage2 - The Design Stage.

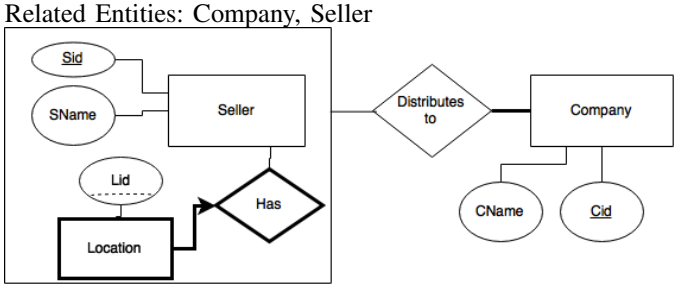
- The relation along with the attributes look like:
  Relation: Buys
  Attributes: UName (account user name) : int, Pid (Part ID) : int, BuysID : int, Date : String
  Relation to user scenario: This relation is to keep track of each unique purchase (BuysID) by a given user with a UName that purchases part with unique Pid at a given date.
  Related Entities: Part, User



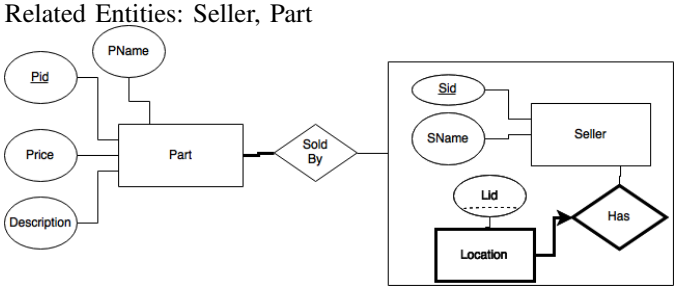
  Relation: Writes
  Attributes: Uid : int, Rid : int, Pid : int
  Relation to user scenario: This relation occurs as described in part 1 when a account user wishes to write at most one review on a given part. Used to keep track of the Uid (user id), and Rid (review id) in relation with the Pid (part id).
  Related Entities: Review, Account User, Part



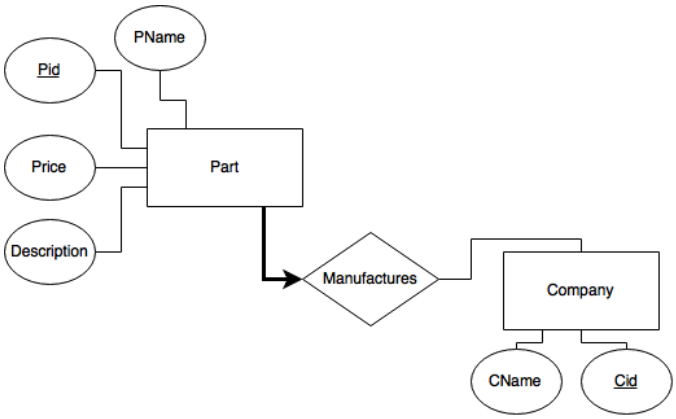
Relation: Distributes To
Attributes: Cid (Company id) : int, Sid (Seller id) : int
Relation to user scenario: When a user wishes to buy a part, they are given a link to the sellers which distribute the part, in order for them to distribute to the users, companies must first distribute the part to each seller. Each company distributes their part to atleast one seller to then sell that part.
Related Entities: Company, Seller



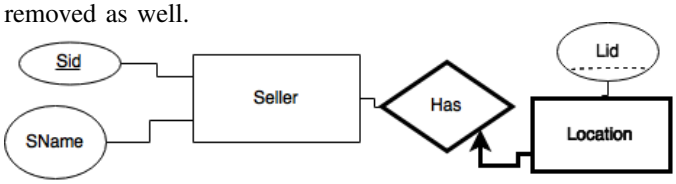
Relation: Sold By
Attributes: Sid : int, Pid : int
Relation to user scenario: Similar to the previous scenario, when an account user makes a purchase they are given a link to the seller of their choosing. This relation relates all sellers to products they sell, and all sellers who sell a certain part.
Related Entities: Seller, Part



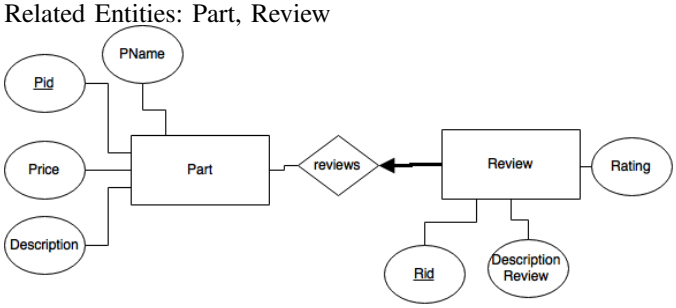
Relation: Manufactures
Attributes: Pid : int, Cid : int
Relation to user scenario: When a company releases a new part, this relation keeps track of how each part is manufactured. Each individual product in the Computer Hardware Database is manufactured by exactly one company. For example: Intel and Kingston cannot both manufacture Intel Core i7 Processors, only Intel can.
Related Entities: Part, Company



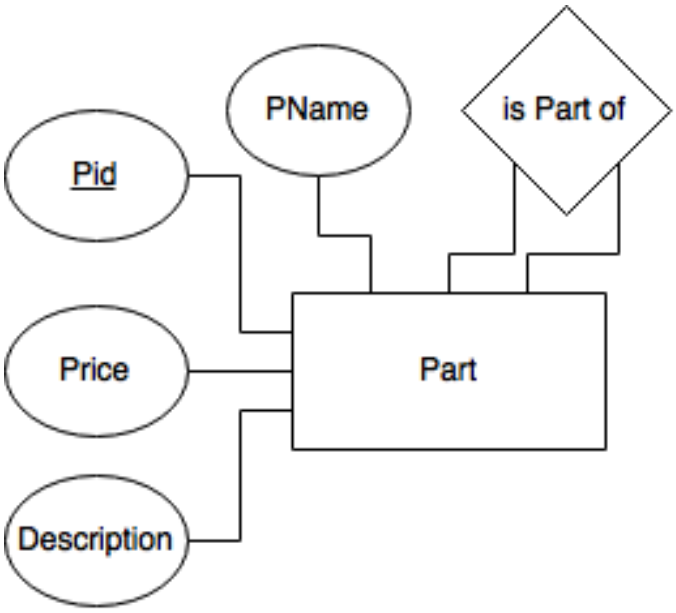
Relation: Has
Attributes: Lid : int, Sid : int
Relation to user scenario: When describing the Seller entity, we decided to describe it in terms of an additional relation and entity within an aggregate. Whenever a seller is called upon within a relation, the seller location will describe that seller within a unique location containing a Lid. This relation is a weak entity in that if a Seller is removed, the location describing that seller will be removed as well.



Relation: Reviews
Attributes: Rid : int, Pid : int
Relation to user scenario: Each review reviews exactly 1 part. This relation keeps track of the review id and the part that it applies to.
Related Entities: Part, Review



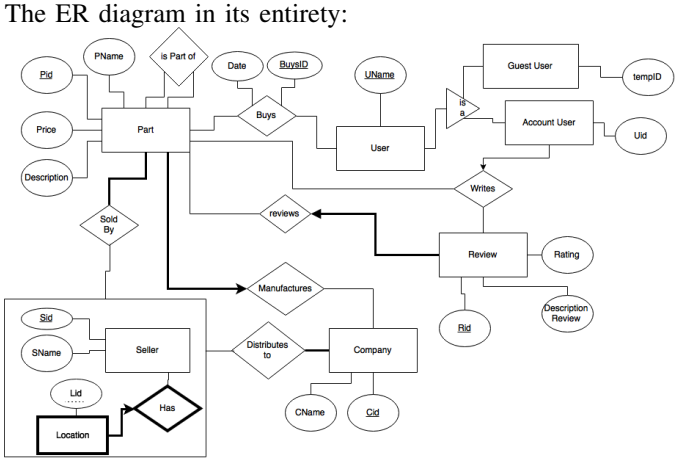
Relation: is Part of
Attributes: Pid1 : int, Pid2 : int
Relation to user scenario: Each part is a component of multiple parts. In a real-world user scenario, computer parts are very complex and made up of multiple individual computer parts. We accounted for this by implementing this relation.
Related Entities: Part



- The description relating this ER part (relation) as it corresponds to one or more user scenario(s): Please insert the description relating this ER part to user scenario(s), in here.

Please repeat that pattern for each relation.

- The ER diagram in its entirety:



- The ER diagram description (corresponding to the user scenarios): Above is our ER diagram depicting numerous user type scenarios that could take place within our computer hardware database. In our ER diagram, we based the different user types as entities themselves, with the addition of a ”Review” and ”Part” entity. Having user types as entities allows us leeway for enacting our user scenarios through the use of relations of those entities. The ER diagram closely follows the user scenarios that were previously described in this report. For example, the ”User” entity ”Buys” a ”Part” that is ”Sold By” a ”Seller”. This scenario is clearly depicted in the ER diagram above and numerous other user scenarios are clearly described as well. With the addition of the entities are their potential attributes and relations to other entities. The ”Writes” relation relates to a ”Review” entity and a ”Account User” entity. The ”Part” entity is tied to six potential relations, which includes ”Buys”,”Has”, ’Manufactures”, ”is Part of”, ”Writes” and ”Sold Buy”. Furthermore, a ”Account User” has the ability to ”Write” a ”Review” or ”Buy” a ”Part”. Lastly, our ”Company” entity has a primary purpose of ”Distributing to” the ”Seller” entity, from whom a ”Part” is ”Sold by”. In regards to the attributes we have chosen for each entity, all entities contain at least one primary key that helps to distinguish themselves from other entities. Overall, we tried to compose a database design that was easy to follow yet completes its task quickly and without problems. All user scenarios are clearly described in this single ER diagram in such a way that connects each of our entities in a logical and reasonable manner.

- The integrity constraints defined in the ER diagram: Each part is sold by atleast one seller, each company distributes to atleast one seller that resides at a particular location, each part is manufactured by exactly one company, a covering constraint indicating that each user type must either be an ”Account User” or a ”Guest User”, a weak entity amongst a ”Location” that only exists if there is a ”Seller” entity and each account user writes at most one review for a given part. Additionally, each ”Review” ”reviews” exactly one ”Part”.
  - Integrity Constraint: Each part is sold by at least one seller.
  - The description and justification of the integrity constraint: The first integrity constraint we included in our ER diagram relates to the ”Part” entity and the ”Sold By” relation. In order to make logical sense of our ER diagram, we created this constraint with the fact that a part is sold by atleast one seller. A single part can be sold by multiple sellers.
  - Integrity Constraint: Each company distributes to at least one seller that resides in a location (aggregate).
  - The description and justification of the integrity constraint: The second integrity constraint we included in our ER diagram relates to the ”Company” entity and the ”Distributes to” relation. We constructed this constraint with the idea that at a Company distributes their manufactured part to atleast one ”Seller”. A company should be able to distribute their part to a collection of sellers.
  - Integrity Constraint: Each part is manufactured by exactly one company.
  - The description and justification of the integrity constraint: The third integrity constraint we included in our ER diagram relates to the ”Part” entity and the ”Manufactures” relation to ”Company’. Each part in the database is manufactured by exactly one company. No identical part is manufactured by more than one company. Each company can manufacture

many parts as well.

- Integrity Constraint: A covering constraint indicating that each user type must either be an ”Account User” or a ”Guest User”
- The description and justification of the integrity constraint: In our model, we have two potential user types that have similar roles but vary in allowed attributes. As compared to account users who carry a unique Uid, guest users carry a tempID that distinguishes them from account users. By implementing an ISA relationship, we can account for both account users and guest users.

- Integrity Constraint: A weak entity amongst a ”Location”. Each seller location only exists if there is a ”Seller” entity.
- The description and justification of the integrity constraint: In order for there to be a seller location, a seller must exist. This indicates that the ”Location” entity is a weak entity in the fact that a location can not exist without a seller. Each location has a seller tied to it and will be deleted if a seller is deleted from the database.

- Integrity Constraint: Each Account User writes at most one review for a given part.
- The description and justification of the integrity constraint: In our computer hardware database design, we decided that an account user type can write a single review on a give computer part. To account for this, we drew an arrow relating the ”Review” and ”Writes” relationship in order to indicate this constraint. An account user should not be able to write multiple reviews on a single computer hardware part. This keeps the database free from user-review spamming and keeps the integrity of the content in our database at a high level.

C. Stage3 - The Implementation Stage.

- The normalization step for the SQL table and the explanations/justification of the normalization step and The normalization steps for each table, along with explanations/justifications and the normalization steps for each table, along with explanations/justifications the SQL Table, including data entries

1) The **Part** Table, including data entries:
PART

Description	Price	Pid	PName
8GB (2 x 4GB) 240-Pin DDR3 SDRAM DDR3 1600	52.99	4355	G.SKILL Ripjaws Series
Devil's Canyon Quad-Core 4.0GHz LGA 1150	339	2445	Intel Core i7-4970k
Silent Computer Case	79.9	6753	Fractal Define S

Normalization Description: The **Part** table is already normalized in 3NF and therefore passes 3NF. There is no need to normalize it because all fields depend on the Pid. The table passes also passes 1NF because each field is atomic and contains no redundant data and nested relations. Lastly, the table passes 2NF because each non-key attribute depends fully on the primary key and not partially. Overall, if we were to change the Pid, this would ultimately impact the Description, Price and PName.

2) The **Buys** Table, including data entries:
BUYS

BuysID	Pid	UName	Date
00010	4355	Techlover15	071415
00011	2445	MechaMan12	072015
00012	6753	RoboLover11	072115

Normalization Description: The **Buys** table is already normalized in 3NF. There is no need to normalize it because all fields depend on the BuysID. The BuysID encompasses information regarding the part, the user and the date purchases. The table also passes 1NF because each field is atomic and contains no redundant data and nested relations. Lastly, the table passes 2NF because each non-key attribute depends fully on the primary key and not partially. Overall, if we were to change the BuysID, the Pid, UName and Date would change value.

3) The **Company** Table, including data entries:
COMPANY

Cid	CName
15210	Kingston
15213	Intel
15217	Asus

Normalization Description: The **Company** table does not need to be normalized because it is already in 3NF. The CName ultimately depends on the Cid. Additionally, the table passes 1NF because there are no layering of relations present nor are the attributes nonatomic. Lastly, the table passes 2NF because the CName is fully functionally dependent on the Cid and not partially. Ultimately, if the Cid changes, the CName will change as well.

4) The **User** Table, including data entries:
USER

UName
Techlover15
MechaMan12
RoboLover11

Normalization Description: The **User** table does not need to be normalized because it contains only a single attribute. It passes 3NF because UName is functionally dependent and determines itself. The table passes 1NF because UName is atomic and the table does not contain multiple layers of relations. Finally, the table passes 2NF because UName is fully functionally dependent on itself and never be partially dependent on itself.

5) The **Guest User** Table, including data entries:
Guest User

tempID	UName
0001	SuperDuperMan231
0002	HelloDatabase12
0003	CSSlam1

Normalization Description: The **Guest User** table does not need to be normalized it is in 3NF. It passes 1NF because the UName and tempID are atomic and do not contain multiple values. Additionally, 2NF is passed because UName (and user functionality) depends on the tempID. Lastly, it passes 3NF because there is no transitive dependency present on the UName attribute on the tempID.

6) The **Account User** Table, including data entries:
Account User

Uid	Uname
1001	Techlover15
2001	MechaMan12
3001	RoboLover11

Normalization Description: The **Account User** table does not need to be normalized it is in 3NF. It passes 1NF because the UName and Uid are atomic and do not contain multiple values. Additionally, 2NF is passed because UName (and user functionality) depends on the Uid. Lastly, it passes 3NF because there is no transitive dependency present on the UName attribute on the Uid.

7) The **Seller** Table, including data entries:
Seller

Sid	SName
05230	Newegg
05240	Evertek
05250	CDW

Normalization Description: The **Seller** table does not need to be normalized it is in 3NF. It passes 1NF because the Sid and SName are atomic and do not contain multiple values. Additionally, 2NF is passed because SName depends on the Sid. Lastly, it passes 3NF because there is no transitive dependency present on the SName attribute on the Sid. The SName is functionally dependent on the Sid.

8) The **is Part of** Table, including data entries:
is Part of

Pid1	Pid2
1101	2101
1102	2102
1103	2103

Normalization Description:
1nf: The attributes Pid1 and Pid2 are not multivalued attributes and have no nested relations.
2nf: Pid1 and Pid2 are both part of the primary key so there are no non-key attributes to be functionally dependent on part of the primary key.
3nf: Again there are no non-key attributes so it is not possible for a non-key attribute to be functionally dependent on another non-key attribute therefore there is no transitive dependency of a non-key attribute on the primary key.

9) The **Review** Table, including data entries:
Review

Rid	Rating	Description Review
00027	9	This RAM is awesome! It outdid my expectations.
00028	8	Intel's CPU i7 Processor works great for my gaming needs. Really good product.
00029	5	I was hoping for a better Motherboard. It really didn't work the way I hoped.

Normalization Description:
1nf: The attributes Rating, Description and Rid are not multivalued attributes and have no nested relations within each other.
2nf: Rating and Description Review are both part of the partial key (Rid) so there are no non-key attributes to be functionally dependent on part of the partial key.
3nf: There are no non-key attributes so it is not possible for a non-key attribute to be functionally dependent on another non-key attribute therefore there is no transitive dependency of a non-key attribute on the primary key.

10) The **Resides** Table, including data entries:
Resides

Sid	Lid
05230	Edison NJ
05240	Temecula CA
05250	Eatontown NJ

Normalization Description:
1nf: The attributes Sid and Lid are not multivalued attributes and have no nested relations within each other.
2nf: Sid and Lid are both used to reference the primary keys from the Seller entity and the Location entity. There are no non-key attributes to be functionally dependent on part of the primary key.
3nf: Again there are no non-key attributes so it is not possible for a non-key attribute to be functionally dependent on another non-key attribute therefore there is no transitive dependency of a non-key attribute on the primary key.

11) The **Location** Table, including data entries:
Location

Lid
Edison NJ
Temecula CA
Eatontown NJ

Normalization Description: The **Location** table does not need to be normalized because it contains only a single attribute. It passes 3NF because Lid is functionally dependent and determines itself. The table passes 1NF because Lid is atomic and the table does not contain multiple layers of relations. Finally, the table passes 2NF because LID is fully functionally dependent on itself and never be partially dependent on itself.

12) The **Distributes to** Table, including data entries:

Sid	Cid
05230	1222228239
05240	1222228240
05250	1222228241

Normalization Description: The **Distributes to** table passes 1NF because all attributes (Sid and Cid) are atomic attributes that do not consist of more than one data entry per tuple. Additionally, the table passes 2NF because there are no non-key attributes and there are no partial dependencies. The only key attributes are Sid and Cid. Finally, the table passes 3NF because there are no transitive dependencies between Sid and Cid.

13) The **Sold By** Table, including data entries:

Sid	Pid
05230	4355
05240	2445
05250	6753

Normalization Description: The **Sold By** table passes 1NF because all attributes (Sid and Cid) are atomic attributes that do not consist of more than one data entry per tuple. Additionally, the table passes 2NF because there are no non-key attributes and there are no partial dependencies. The only key attributes are Sid and Pid. Finally, the table passes 3NF because there are no transitive dependencies between Sid and Pid.

We do not need a table for Manufactures, Has and Writes because the relations are "At most" or "Exactly".

Access Right Levels: Manufacturer, Seller, Guest User, Account User

```
CREATE TABLE Company(
    -> Name VARCHAR(20) NOT NULL,
    -> Cid INT(5) NOT NULL PRIMARY KEY,
    -> CHECK(Cid > 9999 AND Cid < 11000));
```

Name obviously cannot be null as every company needs a name, Same applies to primary key, the checker insures that the ID is in the 10000 -> 11000.
Access Rights: Only manufacturers have the ability to insert into this table, everyone else has read access to everything in the table.

```
CREATE TABLE Seller(
    -> Sid INT(5) NOT NULL,
    -> CHECK (Sid > 0 AND Sid < 10000),
    -> Name VARCHAR(20) NOT NULL,
    -> Lid INT(3) NOT NULL,
    -> PRIMARY KEY (Sid)
    -> FOREIGN KEY (Lid) REFERENCES Location (Lid),
    -> UNIQUE(Name) );
```

Sid is of size 5 because we will not be needing more than that number, we check to make sure the SID is within the range that we set up for those id's, everything else can't be null because it's all vital information, and the primary key is set to Sid. Name is unique because it is not a primary key however no 2 sellers can have the same name.
Access Rights: Only sellers have access to inserting into this table, everyone else has read access on all fields.

```
CREATE TABLE DistributesTo(
    -> Cid INT(5) NOT NULL,
    -> Sid INT(5) NOT NULL,
    -> FOREIGN KEY (Cid) REFERENCES Company(Cid),
    -> FOREIGN KEY (Sid) REFERENCES Seller(Sid));
```

Both id's can't be null, as they are vital to expressing the relation, the foreign keys reference the parent tables which hold references to both id's. no checks to make sure the Cid or Sid fit the conditions already set since they are foreign keys and have to be in the existing tables in the first place in order to be added.
Access Rights: Everyone has read access to this table, Manufacturers have insert access.

-

```
CREATE TABLE Part( Pid INT(5) NOT NULL,
    -> Name VARCHAR(30) NOT NULL,
    -> Price FLOAT(6,2) NOT NULL,
    -> Description VARCHAR (60),
    -> Manufacturer VARCHAR(20) NOT NULL,
    -> PRIMARY KEY (Pid),
    -> FOREIGN KEY (Manufacturer) REFERENCES Company(Name),
UNIQUE (Name));
```

All fields besides description are not null as they are vital, the lengths were selected to fit most similar names of computer hardware. Primary key is set to PID, the name is unique as many parts can't have the same name.
Access Rights: Manufacturers have insert/delete rights on this, however they do not have access to the price field. Sellers have update access to the price field. users have read access of all fields.

```
CREATE TABLE SoldBy(
    -> Pid INT(5) NOT NULL,
    -> Sid INT(5) NOT NULL,
    -> FOREIGN KEY (Pid) REFERENCES Part(Pid),
    -> FOREIGN KEY (Sid) REFERENCES Seller(Sid));
```

Both pid and Sid are foreign keys referencing unique id's in the parts and sellers tables respectively, constraints to make sure the id's are not too big or too small are not necessary as they are covered in the creation of the ID's in the tables being references.
Access Rights: only sellers have insert access to this table, account users have read access, guest users do not.

```
CREATE TABLE User (
    -> Name VARCHAR(20) NOT NULL,
    -> Uid INT(5) NOT NULL,
    -> CHECK (Uid > 0 AND Uid < 6000),
    -> PRIMARY KEY (Uid),
    -> UNIQUE (Name) );
```

Name and ID can't be null as they are vital bits of information, Primary key is set to Uid and all users are identified by a unique user id, however no to users may have the same name, so Name is set to unique. The check is in place to make sure the id is under 6000.
Access Rights: read only access for all parties.

```
CREATE TABLE AccountUser (
    -> Uid INT(5) NOT NULL,
    -> FOREIGN KEY (Uid) REFERENCES User(Uid));
```

Uid can't be empty, and the UID references and assigned user id given to any user who visits the database.
Access rights: Account users only have access to this table.

```
CREATE TABLE GuestUser (
    -> Uid INT(5) NOT NULL,
    -> TempID INT(4) NOT NULL,
    -> FOREIGN KEY (Uid) REFERENCES User(Uid));
```

Uid can't be empty, and all guest users are assigned a temp id which is there to know if a user us a guest or not, which disallows them access to writing reviews or selecting which seller to buy a product from.
Access Rights: everyone has read access to this table.

```
CREATE TABLE Buys(
    -> BuysID INT(7) NOT NULL,
    -> Pid INT(5) NOT NULL,
    -> Uid INT(4) NOT NULL,
    -> Day INT(2) NOT NULL,
    -> Month INT(2) NOT NULL,
    -> Year INT (4) NOT NULL,
    -> PRIMARY KEY (BuysID),
    -> FOREIGN KEY  (Pid) REFERENCES Part(Pid),
    -> FOREIGN KEY (Uid) REFERENCES User(Uid));
```

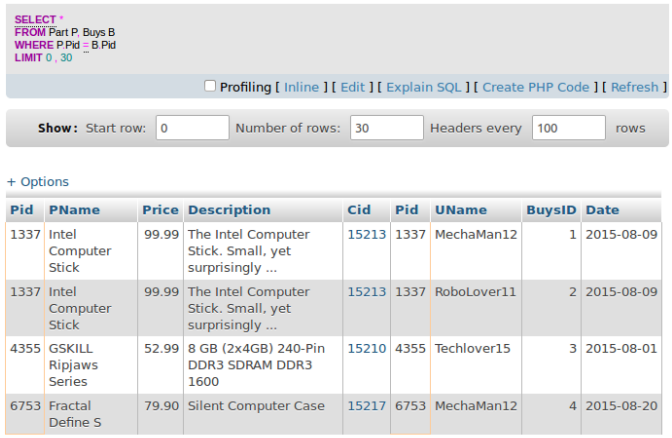
All values can't be null as they are vital, the foreign keys don't need constraints as those constraints must be met in the parent tables in order to exist.
Access Rights: Only users who's uid is equal to a uid in the table would have read access to only rows which match their Uid, Manufactures gain read access to this table.

```
CREATE TABLE Review(
    -> Rating INT(1) NOT NULL,
    -> CHECK(Rating > 0 AND Rating < 6),
    -> Comment VARCHAR(40),
    -> Rid INT(6) NOT NULL,
    -> Pid INT(5) NOT NULL,
    -> Uid INT(4) NOT NULL,
    -> PRIMARY KEY (Rid),
    -> FOREIGN KEY (Pid) REFERENCES Part(Pid),
    -> FOREIGN KEY (Uid) REFERENCES User(Uid));

All values besides comment are necessary and cannot be
null, the check in here makes sure that the rating is
between 1 and 5. each review has a review id that uniquely
identifies it however each review is written by exactly 1
user and applies to exactly one part, so it has 2 foreign
keys that reference those tables.
Access Rights: Users have read access, Account users have
insert access, sellers and manufacturers have read access
only.

CREATE TABLE Writes(
    -> Uid INT(4) NOT NULL,
    -> Rid INT(6) NOT NULL,
    -> FOREIGN KEY (RiD) REFERENCES Review(Rid),
    -> FOREIGN KEY (Uid) REFERENCES User(Uid));

All values are vital and cannot be null, the foreign keys,
again, do not need constraints as they are already checked
in the parent tables.
Access Rights: Read access only to users who uid matches
that on the uid of each row that matches it's own.

CREATE TABLE IsPartOf(
    -> SmallPid INT(5) NOT NULL,
    -> BigPid INT(5) NOT NULL,
    -> PRIMARY KEY (SmallPid).
    -> PRIMARY KEY (BigPid),
    -> FOREIGN KEY (Pid1) REFERENCES Part(Pid),
    -> FOREIGN KEY (Pid2) REFERENCES Part(Pid));

Both parts must exist, again, no checks are necessary to
make sure the id's fit pre-defined parameters.
Access Rights: Read access for all, manufacturers have
insert access.

CREATE TABLE Location (
    -> Lid INT (3) NOT NULL,
    -> PRIMARY KEY (Lid));

Location cannot be null, as it identifies a unique
location. no check here as location can be any 3 digit
number.
Access Rights: read by all

CREATE TABLE Resides(
    -> Lid INT(3) NOT NULL,
    -> Sid INT(5) NOT NULL,
    -> FOREIGN KEY (Lid) REFERENCES Location(Lid),
    -> FOREIGN KEY (Sid) REFERENCES Seller(Sid));

Each seller resides in a location, so each attribute cannot
be null and the foreign keys reference their parent tables
location and sellers.
Access Rights: Read by all

//////////////////// TRIGGERS ////////////////////////

CREATE TRIGGER Manufacturer_Delete
ON Company
AFTER DELETE
AS
BEGIN
DELETE FROM Part WHERE Cid = (SELECT (Cid) FROM DELETED)
END
deletes all parts that have a company id matching that of
the deleted company.
```

D. Stage4 - User Interface.

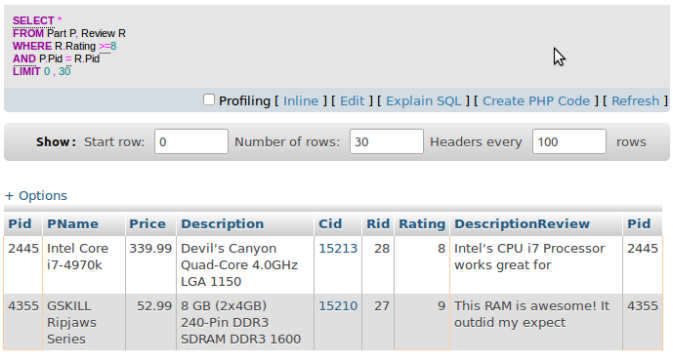
- The the first SQL statement used to query the data: Here is a simple query to start off:
  SELECT *
  FROM Part P
  WHERE P.PName = ”Intel Computer Stick”
  AND P.Pid = 1337

- The tables taking part in the query statement (table names and headers):
  - The name of the first table taking part in the query: Part table
  - The header of the table (all attributes): Pid, PName, Price, Description, Cid
  - The attributes of the table taking part in the query: PName and Pid
- The the second SQL statement used to query the data:
  SELECT *
  FROM Part P, Buys B
  WHERE P.Pid = B.Pid



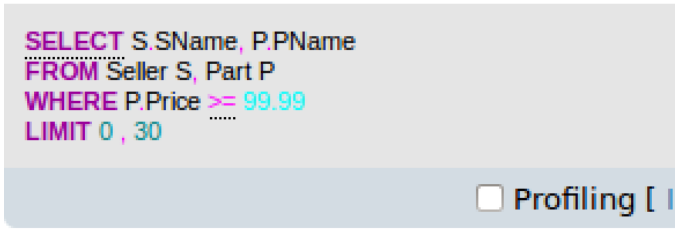
Pid	PName	Price	Description	Cid	Pid	UName	BuysID	Date
1337	Intel Computer Stick	99.99	The Intel Computer Stick. Small, yet surprisingly ...	15213	1337	MechaMan12	1	2015-08-09
1337	Intel Computer Stick	99.99	The Intel Computer Stick. Small, yet surprisingly ...	15213	1337	RoboLover11	2	2015-08-09
4355	GSKILL Ripjaws Series	52.99	8 GB (2x4GB) 240-Pin DDR3 SDRAM DDR3 1600	15210	4355	Techlover15	3	2015-08-01
6753	Fractal Define S	79.90	Silent Computer Case	15217	6753	MechaMan12	4	2015-08-20

- The tables taking part in the query statement (table names and headers):
  - The name of the tables taking part in the query: Part and Buys table
  - The header of the table (all attributes): Part: Pid, PName, Price, Description, Cid
    Buys: Pid, UName, BuysID, Date
  - The attributes of the table taking part in the query: Pid
- The the third SQL statement used to query the data:
  SELECT *
  FROM Part P, Review R
  WHERE R.Rating ¿= 8
  AND P.Pid = R.Pid



Pid	PName	Price	Description	Cid	Rid	Rating	DescriptionReview	Pid
2445	Intel Core i7-4970k	339.99	Devil's Canyon Quad-Core 4.0GHz LGA 1150	15213	28	8	Intel's CPU i7 Processor works great for	2445
4355	GSKILL Ripjaws Series	52.99	8 GB (2x4GB) 240-Pin DDR3 SDRAM DDR3 1600	15210	27	9	This RAM is awesome! It outdid my expect	4355

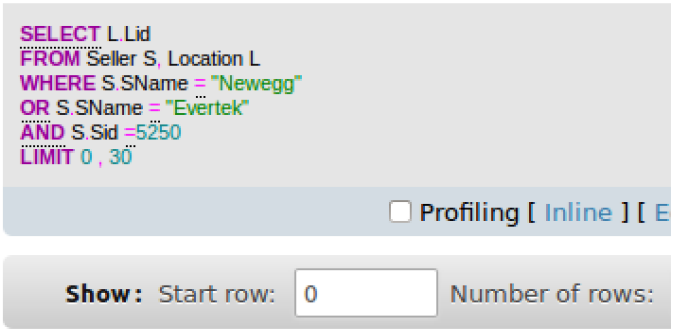
- The tables taking part in the query statement (table names and headers):
  - The name of the tables taking part in the query: Part and Review table
  - The header of the table (all attributes): Part: Pid, PName, Price, Description, Cid
    Review: Pid, Rid, Rating, Review Description
  - The attributes of the table taking part in the query: Pid, rating
- The the fourth SQL statement used to query the data:
  SELECT S.SName, P.PName
  FROM Seller S, Part P
  WHERE P.Price ¿= 99.99



+ Options

SName	PName
Newegg	Intel Computer Stick
Evertek	Intel Computer Stick
CDW	Intel Computer Stick
Newegg	Intel Core i7-4970k
Evertek	Intel Core i7-4970k
CDW	Intel Core i7-4970k

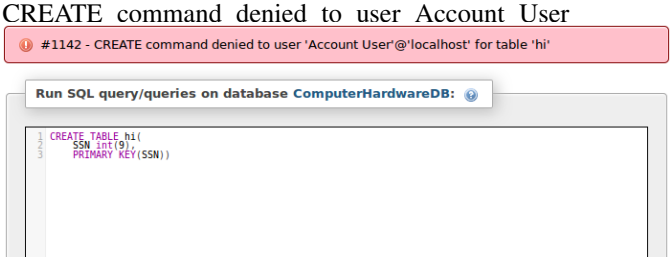
- The tables taking part in the query statement (table names and headers):
  - The name of the tables taking part in the query: Seller and Part tables
  - The header of the table (all attributes): Part: Pid, PName, Price, Description, Cid
    Seller: Sid, SName
  - The attributes of the table taking part in the query: SName and PName
- The the fifth SQL statement used to query the data:
  SELECT L.Lid
  FROM Seller S, Location L
  WHERE S.SName = ”Newegg” OR S.SName = ”Evertek” AND S.Sid = 5250



+ Options

Lid
9997 Rose Hills Road Whittier, CA 90601
43195 Business Park Drive, Temecula, CA 92590
260 Industrial Way West #1, Eatontown, NJ 07724

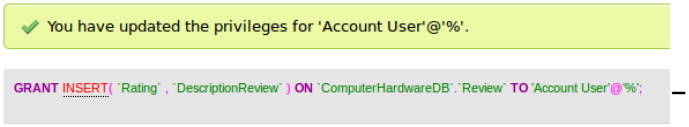
- The tables taking part in the query statement (table names and headers):
  - The name of the tables taking part in the query: Seller and Location table
  - The header of the table (all attributes):
    Location: Lid
    Seller: Sid, SName
  - The attributes of the table taking part in the query: Lid
- The error messages popping-up when users access and/or updates are denied (along with explanations and examples):
  - The first error message:
    CREATE command denied to user Account User



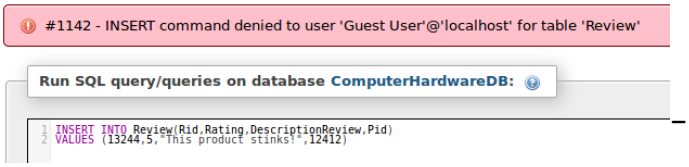
  - The error message explanation (upon which violation does it take place): The user does not have the access

rights for creating a table.

- The error message example according to user(s) scenario(s): This account user accessing the Computer Hardware DB tried creating a table in the database. Account users, however, do not have access rights to do so. This account user must have mixed up his command with INSERT, a command which Account users are allowed access when inserting a rating and description review for the Reviews entity.
  Example:

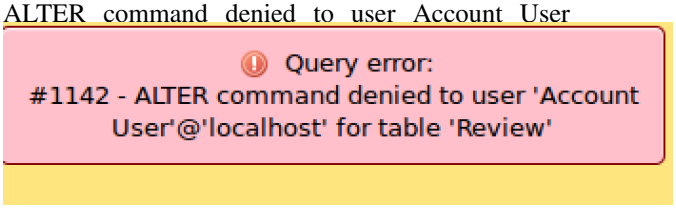


- The second error message:
  INSERT command denied to user Guest User



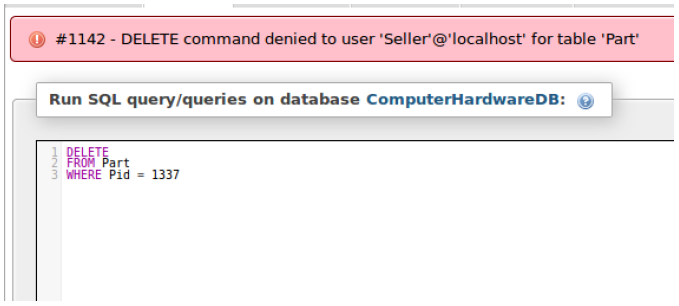
- The error message explanation (upon which violation does it take place): The user does not have the access rights for inserting into the database.
- The error message example according to user(s) scenario(s): This guest user accessing the Computer Hardware DB tried inserting into the Review entity. Guest users, however, do not have access rights to do so. Guest user cannot write reviews or rate reviews like account users.

- The third error message:
  ALTER command denied to user Account User



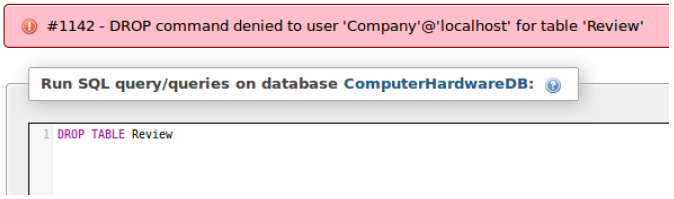
- The error message explanation (upon which violation does it take place): The user does not have the access rights for altering a table in the database.
- The error message example according to user(s) scenario(s): This account user accessing the Computer Hardware DB tried to alter a table in the database. Account users, however, do not have access rights to do so. Account user cannot alter pre-existing attribute specifications in a table.

- The fourth error message:
  DELETE command denied to user Seller User



- The error message explanation (upon which violation does it take place): The user does not have the access rights for deleting information from a table in the database.
- The error message example according to user(s) scenario(s): This seller accessing the Computer Hardware DB tried to delete a Pid from a table in the database. Seller users should not have access rights for deleting product information on a company's part. Only Company users should be able to remove their own information on a given part.

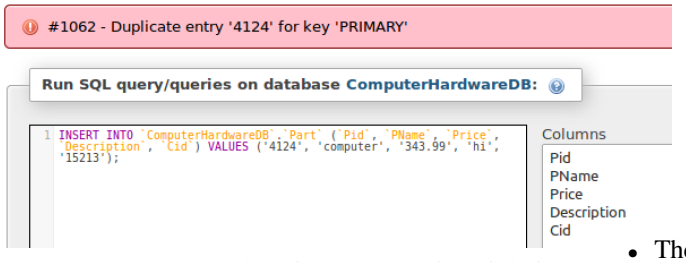
- The fifth error message:
  DROP command denied to user Company User



- The error message explanation (upon which violation does it take place): The user does not have the access rights for dropping an entire table from the database.
- The error message example according to user(s) scenario(s): This company user accessing the Computer Hardware DB tried to drop the Review table from the database. Company users, as well as all users of the database (with the exception of the DBA) should not have access rights for dropping entire tables from the database. The company user may have wanted to delete a review on their product that an account user did not like. Although it is frustrating to see bad reviews on a product one may have created, a prestigious company such as Intel should not take it to heart and rather learn from their potential falters. Ultimately, company users cannot and should not drop review information on their products.

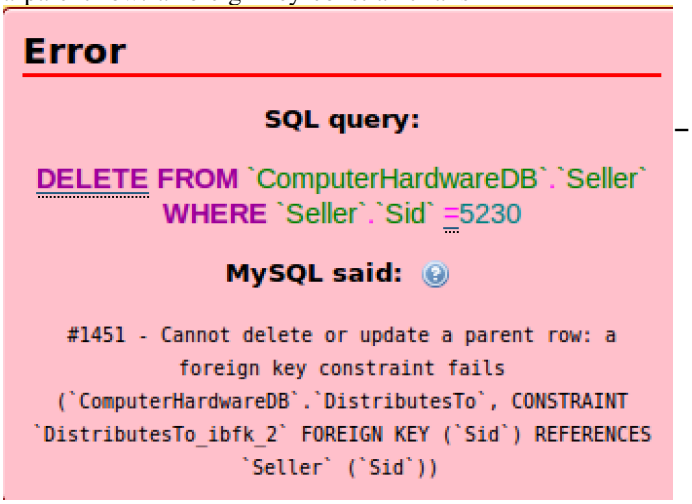
- The error messages corresponding to the integrity constraints violations (along with explanations and examples).

  - The first error message: Duplicate entry '4124' for key 'PRIMARY'



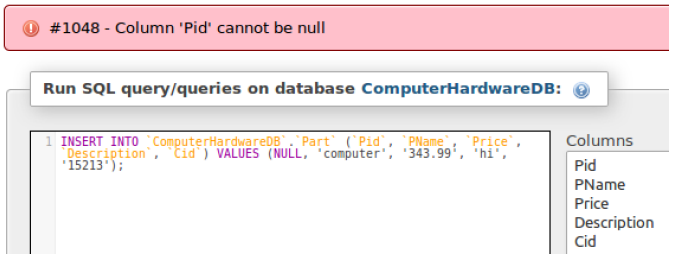
- The error message explanation (upon which violation does it take place): When trying to INSERT a duplicate primary key for Pid, the system stops the command with a duplicate primary key constraint.
- The error message example according to user(s) scenario(s): In order to ensure that changes made to the database do not result in loss of data and consistency, our database holds PRIMARY keys. These keys (such as Pid) allow our products in Part to be unique and have individual characteristics from all parts being added to the database. In the case if a seller tried to INSERT a duplicate part by accident in their retail system, the database system would catch it and save the seller time and money trying to find the root cause that is throwing off their inventory calculations. Having duplicate part ids can most definitely throw off a inventory database, and it is accounted for in our database.

- The second error message: Cannot delete or update a parent row: a foreign key constraint fails

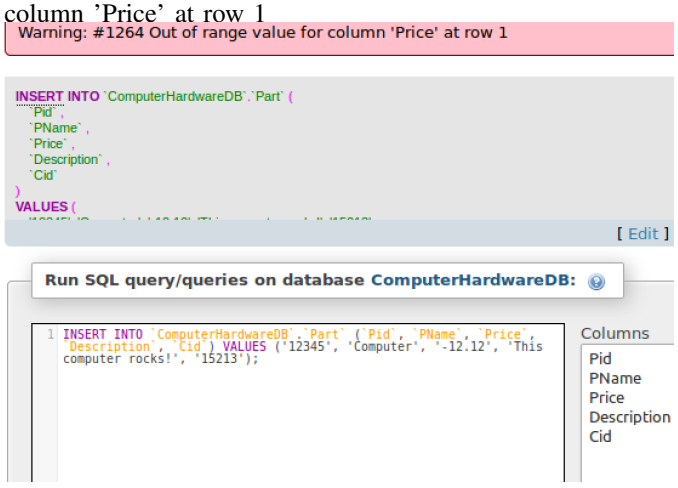


- The error message explanation (upon which violation does it take place): When trying to DELETE a seller tuple from the database, a FOREIGN key constraint is thrown.
- The error message example according to user(s) scenario(s): In order to ensure that changes made to the database do not result in loss of data and consistency, our database holds FOREIGN keys. These keys allow our tables and entities within the database to be interconnected. In this example regarding the error message above, a person using the database, most likely a seller user, tried to delete their seller information from the database. This violates the database because information regarding seller id (Sid) is already linked to Companies through relation Manufactures as well as Part through relation SoldBy. This would throw off the database entries for multiple entities and relationships since they are all connected through use of foreign keys.

- The error messages corresponding to the data range constraints violations (along with explanation).
  - The first error message: Column 'Pid' cannot be null



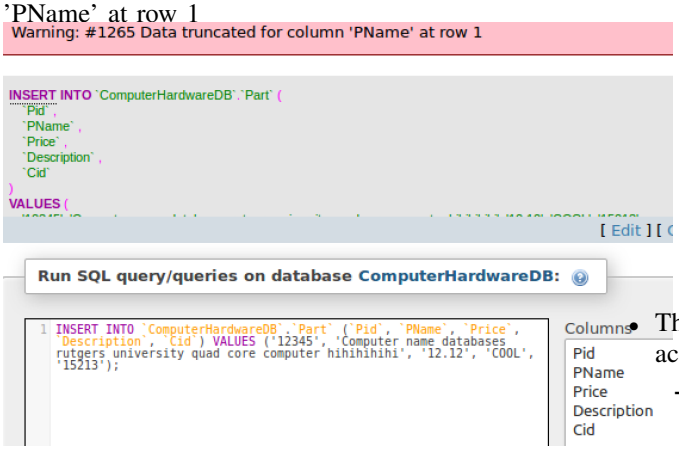
  - The error message explanation (upon which violation does it take place): For each PRIMARY key in the database, the value of it cannot be NULL. This would allow no uniqueness for an entity if it were given a NULL value.
  - The error message example according to user(s) scenario(s): An example where this error message might arise could be when a seller does not know the product id (Pid) initially but wants to insert its price, description and name. The seller cannot include NULL as information for Pid because it is a PRIMARY key. There would be no way of referencing that part if it was given a NULL value.
  - The second error message: Out of range value for column 'Price' at row 1



  - The error message explanation (upon which violation does it take place): For INT and FLOAT values in the database, they must be regarded as UNSIGNED values. There should be no negative values inserted into the database.
  - The error message example according to user(s) scenario(s): Most likely, the seller selling this part made a typo in the computer hardware database. A

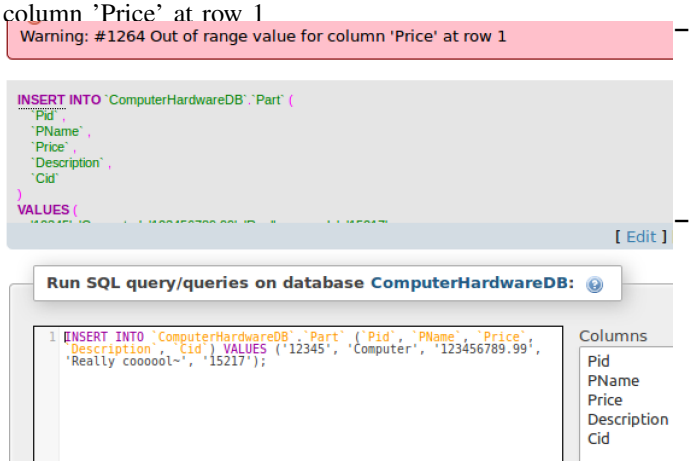
price cannot and should not be listed as a negative float value.

- The third error message: Data truncated for column 'PName' at row 1



- The error message explanation (upon which violation does it take place): For each product name in the Part entity, there is a limit to how long a PName can be. The error is throw because the PName is longer than varchar(30).
- The error message example according to user(s) scenario(s): For all varchar fields in the database, there is a certain limit to the amount of characters one can type for that attribute. This data-range constraint is to allow concise descriptions of each product name. We want to create a simple yet effective environment for our account users and guest users, and by making product descriptions clear and concise, we can achieve this. The database will automatically truncate the data if too long, hinting this to the seller/company to fix and alter immediately.

- The fourth error message: Out of range value for column 'Price' at row 1

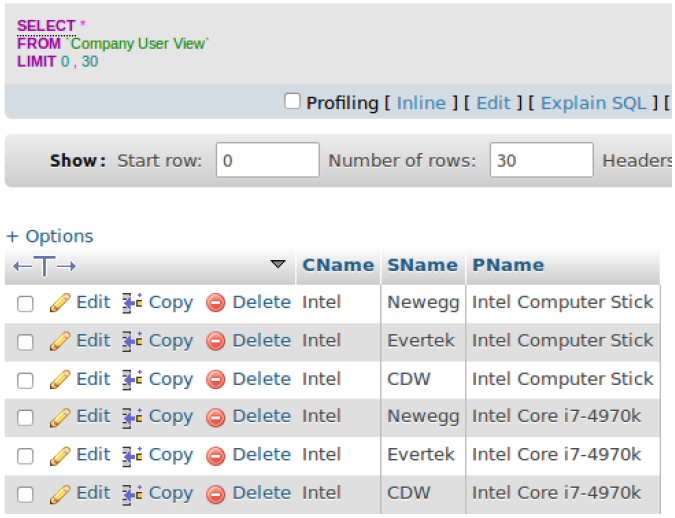


- The error message explanation (upon which violation does it take place): The price of a product cannot exceed one million dollars in our database. Data that is larger than the specified data type will cause issues if not notified and dealt with immediately. We have accounted for this by adding a data-range constraint for the price field in the part entity through use of float(6,2).
- The error message example according to user(s) scenario(s): An example where this error might be thrown is when a seller lists the price of a part to be that greater than 999999.99. We do not want to deal with products greater than this value because it is simply a lot of money to deal with and there lies a lot more pressure on the database to keep the integrity and security of each transaction. If our database were to go through increased enhances involving data and transaction security, we would most definitely deal with such a fit.

- The header of the views created in order to facilitate data accesses:
- The first view created:
  Selects to view the part name, description, price, rating and description review.

- The view justification: This view was created with the Account User in mind. In order to make it simple for account users to make their purchase decisions, we created a view that shows all information regarding each part as well as reviews and ratings on that part.
- The tables taking part in the view and the specific attributes taking part in the view:
  * The table names: Part and Review tables.
  * The table headers (all attributes):
    Part: Pid, PName, Description, Price, Cid
    Review: Rid, Rating, DescriptionReview, Pid
  * The table attributes that take part in the view:
    Part: PName, Description, Price
    Review: Rating, DescriptionReview
- The SQL statement used for creating the view:
  SELECT P.Pname, P.Description, P.Price, R.Rating, R.DescriptionReview
  FROM Part P, Review R

– The second view created:
Selects to view all of Intel's seller distributors of their products.



	CName	SName	PName
Edit Copy Delete	Intel	Newegg	Intel Computer Stick
Edit Copy Delete	Intel	Evertek	Intel Computer Stick
Edit Copy Delete	Intel	CDW	Intel Computer Stick
Edit Copy Delete	Intel	Newegg	Intel Core i7-4970k
Edit Copy Delete	Intel	Evertek	Intel Core i7-4970k
Edit Copy Delete	Intel	CDW	Intel Core i7-4970k

– The view justification: This view was created with the Company User in mind. In order to keep track of their distribution of products, companies can create a view that allows them to see products they are currently distributing to different sellers.
– The tables taking part in the view and the specific attributes taking part in the view:
  * The table names: Company, Seller and Part tables.
  * The table headers (all attributes):
    Company: Cid, CName
    Part: Pid, PName, Description, Price, Cid
    Seller: Sid, SName
  * The table attributes that take part in the view:
    Company: CName
    Part: PName
    Seller: SName
– The SQL statement used for creating the view:
SELECT C.CName, S.SName, P.PName
FROM Company C, Seller S, Part P
WHERE C.CName = "Intel" AND C.Cid = P.Cid